Time Series Deinterleaving with Application in Malicious Domain Detection

Amir Asiaee T.
Mathematical Biosciences Institute
Ohio State University
Columbus, USA
asiaeetaheri.1@osu.edu

Hardik Goel
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address

Shalini Ghosh
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address

Vinod Yegneswaran
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address

Arindam Banerjee
Department of Computer Science
University of Minnesota
Minneapolis, USA
banerjee@cs.umn.edu

***Abstract*—This is the abstract.**
***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

Deinterleaving temporal data streams is a general machine-learning problem with important applications to security and privacy. Specifically, interleaved network data streams are a common occurrence in cyber-threat monitoring which complicates many analyses. In many instances, the individual stream identifiers are unavailable due to technical challenges, such as the vantage point of the data collector or are intentionally supressed to protect the privacy of the users in the network.

For example, consider packet traces collected in a local area network, where the source IP addresses are removed or data collected from the external-facing interface of a proxy server or NAT firewall where individual client identifiers are unavailable. Detecting anomalous behavior, especially stealthy and low-volume attack patterns, in these aggregated noisy streams is significantly more challenging than in a traditional deinterleaved setting.

In this paper, we discuss a variant of this problem, i.e., deinterleaving client request streams from recursive DNS resolvers to mine threat intelligence. Such DNS data streams are shared among Internet service providers (ISPs) through mediums such as the Security Information Exchange (SIE) and are a valuable source of intelligence to the cybersecurity community. Here, the individual client requests to the recursive DNS resolver are typically suppressed and what we have are inter-resolver communications (i.e, communications between the recursive resolver and the root server, TLD servers and other secondary resolvers). We are interested in the application of advanced machine-learning techniques to automate the extraction of malware domain groups [] from such resolver streams.

Identify applicable funding agency here. If none, delete this.

Malware infections while browsing the Internet have become very prevalent and occur due to various reasons such as drive-by exploits, phishing attacks etc. In a typical infection, the user starts from a landing page and then goes through a sequence of seemingly harmless intermediate websites, until reaching a site that contains the malicious exploit that harm the user by installing malware or stealing private data. The intermediate sites are typically redirection chains implemented in JavaScript for the purpose of obfuscation. Even though many landing and exploit websites are continously identified and blacklisted, thousands of new malicious domains emerge daily. However, pieces of the redirection infrastructure get reused across campaigns and thus the actual sequence of websites traversed by the user contains information that may help in quickly identifying new exploit sites.

When a user makes a browser request to visit a website, it first resolves the domain name by asking its recursive resolver. If the answer for the query is cached by the resolver the answer is immediately provided to the client. Otherwise, it initiates a set of recursive queries, leading to the final queried website's IP address. Each webpage may have several embedded objects from many domains leading to a sequence of domain lookup requests emanating from the client. Tracking the set of DNS requests made by each client is thus a useful means to identifying new and emergent malware infection sequences. However, to protect user privacy ISPs typically only capture data from the external facing interface of the recursive resolver, effectively suppressing the individual client stream identifiers. As there are hundreds of users making requests at the same period of time, and all of these requests are pushed to a single queue of a local DNS resolver, we cannot tell apart individual user's sequences of requests and perfectly deinterleaving all requests for deanonymization purposes is impossible. However, our objective is not deanonymization, but rather extraction of malware domain sequences which are observed repeatedly across resolvers. We believe that advanced machine learning

strategies could be in such selective deinterleaving of DNS time-series for the extraction of malware domain groups.

Prior Work. To the best of our knowledge deinterleaving has not been applied to DNS resolver queue's data. Some earlier work [6] investigates the use of a sliding window approach to identify new malicious domains by exploring the domains that typically form neighbors of known malicious domains in the resolver queue, while ignoring the actual sequential information. The challenges of applying existing deinterleaving methods to DNS data is twofold. First, most of the methods has been designed for deinterleaving Markov chains [2], [12]–[14] and HMMs [10], and as we will discuss in Section ?, the dynamic of submitting new queries to the local resolver is more complicated than simple Markov chain or HMM. Moreover, the state space of the models and number of sequence sources are very small in previous work applications [10], [12], while in our application, huge number of websites explodes the size of state space and also tens of users may be active in a network simultaneously. Because of the nature of our dataset, we need to use tools other than those adopted in literature [3], [4], [10], [12].

Another very useful model for time-series is Recurrent Neural Networks (RNN). Recently, RNNs and their variants (Gated Recurrent Units (GRUs) [5], Long Short-Term Memory (LSTMs) [8]) have seen a lot of success in modeling time-series in multiple domains [1], [7], [15]. However, to our knowledge even simple RNN tools have not been applied to the deinterleaving problem. Using RNN-type tools for deinterleaving mixed DNS request logs is a completely unexplored area. Motivated by the power of RNNs to model non-linear dependencies, we seek to apply RNNs to such data and start a new direction of work towards identifying newer malicious domains more efficiently.

Contributions. This paper presents a preliminary exploration of the utility of various machine-learning models to address the time series deinterleaving problem for malware domain group extraction. Specifically, we present a model for DNS request generation and resolver-sequence interleaving and evaluate the utility of various inference strategies on sythetic examples including Viterbi, RNNs and LSTMs finding that LSTMs outperform simple RNNs and Viterbi. Extending this analysis to real and large-scale datasets is future work.

II. Problem Formulation

In this section we first discuss the nature of resolver's data. Then we propose a generative model that captures the essence of our understanding of per-user request generation. Finally we describe the process which interleaves the queries generated by different users and form the final resolver queue.

A. Nature of Data

Here we provide an example that summarize the details of the nature of real data based on which we present our generative model in the following sections. A user starts checking a page e.g., *a.com*. While launching the webpage, many queries are being generated from different components of that browsed webpage: a.com, ad1.com, audio1.org. We call this surge of query generation an *episode*, which happens during *webpage* loading. Other than the page's domain name a.com, other requested domains may be related to the content of the webpage or personalized for the user. Next, the user moves to another sub-domain of a.com or completely opens a new website in both cases another episode starts. A same process generates query sequences for other users. A second user generates: b.com, ad2.com and after the interleaving we may observe the following sequence in the resolver: b.com, a.com, ad1.com, ad2.com, audio1.org. We call this process *request interleaving*. Our goal is to entail each user's query sequence from the mixed resolver queue.

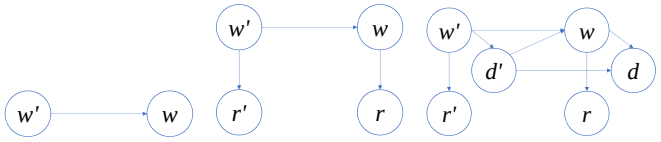
B. User's Request Generation Model

Interleaving process occurs on top of the users' browsing process. The *browsing process* of a user can be modeled as simple as a Markov chain (MC) of webpages, an HMM, or an HsMM model. Figure 1 illustrates these three different user model.

In this section we present a most detailed model that explains user requests generation based on the process elaborated in Section II-A. We use Hidden Semi-Markov Model (HsMM) as our user browsing model, MC and HMM are special cases of this process. The hidden layer of the HsMM consists of random variables W representing the browsed webpages. Note that pages are hidden because what we see are only the requests, e.g., if we just look at the queue of the example of Section II-A maybe the main pages are b.com which has the link to a.com and ad1.com, both queried after it. Therefore, we don't know what page was initially loaded and generated its following queries in the queue.

The page transition matrix is different for each user and is represented by the matrix $\mathbf{P}_u$. The duration random variable D of the hidden layer is the number of queries that are going to be requested by the page. And finally, the observed state of the HsMM is the domain name request R which will be put in the resolver queue. Note that the time between subsequent browsed pages (which is equal to the time spend in a page before moving to the next one) in reality is different from the duration parameter in our model. In real world data, each user spends an interval on a page but in our model since we are only interested in the order of queries, we only count the number of requests that the page will query from the resolver and represent it by random variable D. So the duration parameter D represent the number of requests that are remained to be submitted by the current page.

Fig 1c shows the details of the model and Table I summarizes the model parameters. $\mathbf{O}_u(w, r)$ is the probability of submitting (outputting/observing) request r on the webpage w, for the user u. Conditional probabilities of the model are as follows:



(a) Markov Chain (b) HMM (c) HsMM

Fig. 1: User's browsing models.

Symbol	Explanation
W	RV for the webpage
D	RV for the number of requests to be issued on a page
R	RV for the issued DNS request
m	Number of users
n	Total number of pages
q	Maximum number of requests per page
$\mathbf{P}_u \in \mathbb{R}^{n\times n}$	Webpage transition matrix of the user u
$p_w(d), d \in [q]$	Distribution of number of requests d on the page w
$\mathbf{O}_u \in \mathbb{R}^{n\times n}$	Output distribution matrix of the user u

TABLE I: Summary of the model parameters and random variables (RV). For each random variable the corresponding small letter represents a realization. Note that W and D depend on the user but to avoid cluttering we omitted the index u.

$$\begin{aligned}\mathbb{P}_u(w|w',d') &= \begin{cases}[\mathbf{P}_u]_{w'w} & d'=1\\ \delta(w,w') & d'>1\end{cases},\\ \mathbb{P}_u(d|w,d') &= \begin{cases}p_w(d) & d'=1\\ \delta(d,d'-1) & d'>1\end{cases},\\ \mathbb{P}_u(r|w) &= [\mathbf{O}_u]_{w,r},\end{aligned} \tag{1}$$

The duration parameter can not be zero, when $d = 1$ the page's last request is submitted and the user moves to another page which resets the duration using $p_w(d)$. The duration probability $p_w(d)$ determines the number of requests that the webpage w will query and is independent of user u.

C. Resolver's Sequence Interleaving Model

As mentioned in Section II-A, we have access only to the resolver data queue where users enqueued their domain name requests. Here we assume that there is a Markov chain governing the "turn" of users. Let's assume that we have m users. We name the transition matrix of the user's Markov chain as $\mathbf{A} = [\alpha_{ij}] \in \mathbb{R}^{m\times m}$. Therefore the probability of user j generating the t-th request given the current user as i is $\mathbb{P}(U(t) = j|U(t-1) = i) = \alpha_{ij}$. The random variable $U(t) \in [m]$ represent the active user that has generated the tth request of the resolver queue.

A simplification of the transition matrix that has been use in literature is $\forall i,j : \mathbb{P}(U(t) = i|U(t-1) = j) = \alpha_i$, where the whole dynamic of the system can be represented by a single users *shares vector* $\boldsymbol{\alpha}$ instead of the *users transition matrix* $\mathbf{A}$.

To distinguish each user's corresponding HsMM random variable in the interleaving process we use both user index and time index. For example, $W_k(t)$ is the user k's current webpage. Note that here the time is different from the real world time and HsMM duration that discussed in Section II-B. Time here is just an index into the resolver's sequence of queries. For example, $W_k(t)$ shows the webpage of user k when the tth request was submitted to the resolver. Note that, the tth request may have been generated by any of the users not only k.

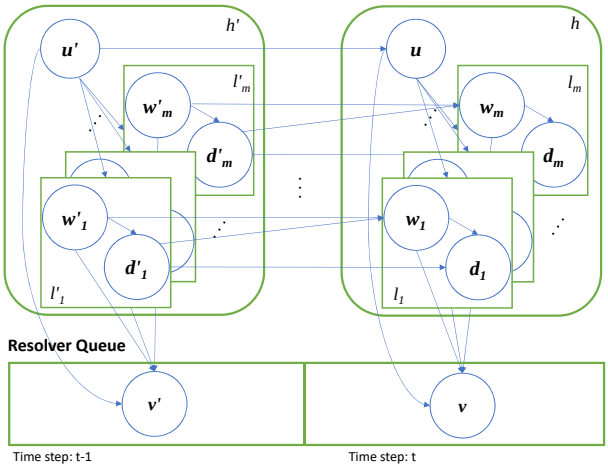


Fig. 2: Illustration of the interleaving process.

To elaborate the interleaving process, we describe it as an Augmented Hidden Markov Model (AHMM), where the hidden states are augmented state, i.e., combination of variables [12]. To make the equations more readable, we lump together the variables corresponding to each user and make the following lumped variable $L_k(t) = (W_k(t), D_k(t))$ and the hidden state of the HMM becomes $H(t) = (L_1(t), \ldots, L_m(t), U(t))$ which is a $2m + 1$ dimensional vector. Fig 2 illustrates the interleaving process that leads to sequence generation. To avoid cluttering in the following we assume $u(t-1) = u'$ and $u(t) = u$ which means that users u' and u are active at time steps $t-1$ and t respectively.

At the time step t, user $u(t) = u \in [m]$ generates the request $v(t)$ which is the observed (visible) variable of the HMM. The request $v(t)$ is determined by the next request of the user in its HsMM model, i.e., $r_u(t)$. Therefore, the emission probability of the AHMM is:

$$\mathbb{P}(V(t) = v(t)|H(t) = h(t)) = \mathbb{P}_u(r_u(t)|w_u(t)) = \mathbf{O}_u(w_u(t), r_u(t)).$$

Now we derive the entries of the transition probability matrix of the AHMM:

$$\mathbb{P}(H(t)|H(t-1)) = \alpha_{u'u}\prod_{k=1}^{m}\mathbb{P}(l_k(t)|l_k(t-1), u), \tag{2}$$

In the case of $k \neq u$ the user k is not active, i.e., stalled. Substituting the probability distributions from (1), we get:

$$\mathbb{P}(l_k(t)|l_k(t-1), u) = \begin{cases}k \neq u & \delta(w,w')\delta(d,d')\\ k = u & \begin{cases}d=0 & p_w(d)[\mathbf{P}_u]_{w'w}\\ d>0 & \delta(d,d-1)\end{cases}\end{cases} \tag{3}$$

Symbol	Explanation
L	The lumped random variable $L = (W, D)$.
H	The hyper-hidden state of the HMM $H = (L_1, \ldots, L_m, U)$.
V	The visible state of the HMM which is the requested DN.

TABLE II: Summary of the augmented variables. For each random variable the corresponding small letter represents a realization.

III. Deinterleaving Methods

In deinterleaving problem, given $\{v(t)\}_{t=1}^{T}$ we are interested in inferring $\{u(t)\}_{t=1}^{T}$. In other words, we want to find the users initiated each request from the sequence generated by the interleaving process described in Section II-C.

We present two candidate approaches for inference. One is based on reducing the interleaving process to a AHMM as discussed in Section II-C. This approach has been used for deinterleaving of Markov chains with small number of chains (users) and state space [12]. Next, we propose to deinterleave using Recurrent Neural Network (RNN) and Long-Short Term Memory (LSTM) which has been performed successfully in many time-series analysis task recently [5], [8].

A. Inference on Augmented HMM

As discussed in Section II-C, we can consider the whole interleaving process as a huge hidden Markov model, AHMM. Then using well-known learning method like EM, we can learn the parameters of the AHMM and finally perform Viterbi inference to determine the most probable hidden (augmented) states $h(t)$ from which we can extract the most probable user $u(t)$. The main difficulty of applying this framework is that the state space of hidden variable, Figure 1, is very large. More specifically, there are $m(nq)^m$ possible states of h and as we increase number of webpages n or users m the state space grows exponentially. The huge state space, makes the inference and learning very hard and as we show in Section IV-A for synthetic experiments, even when the model parameters are known, deinterleaving performs (using Viterbi coding) poorly.

B. Inference using RNN

A popular approach for modeling the time series data is Recurrent Neural Network (RNN) [11]. Given input v_t and hidden state h_{t-1}, RNN computes next hidden state representation h_t and the output u_t using the following recurrent relationships

$$h_t = f(W_v v_t + W_h h_{t-1} + b) \tag{4}$$
$$u_t = W_u h_t \tag{5}$$

where W_v, W_h, W_v and b are the network parameters, and $f()$ is some non-linear function. An example of f could be a sigmoid $f(z) = \sigma(z) = 1/(1 + \exp(-z))$ or rectified linear unit $f(z) = \max(0, z)$.

For our specific problem of deinterleaving, an RNN can be used by posing it as a multi-class classification problem, where the input is the observed webpage and the output will be the identified user who requested that webpage. Specifically, Each data instantiation consists of a sequence of user-request pairs, i.e., $(u(t), v(t))$. This represents who was the user at a given time t and what request was produced by that user. Both the user and the request are represented by an integer. The RNN is unrolled for the entire length of one sequence. The users and request integers are converted to one-hot encoding to enable learning. Thus if there are b possible web pages, the requests become b-dimensional vectors and for m users, it becomes an m-dimensional vector. The request vectors are fed as input to the RNN model, while the output is the corresponding user at each time-step. The RNNs m-dimensional output is passed through a softmax layer to convert it into probabilities and the user with higher probability is compared against the ground truth. Performance is measured in terms of accurately identifying the user at each instant.

A common approach to solving the optimization problem is to randomly initialize network parameters W_v, W_h, W_u and apply stochastic gradient descent (SGD) (for RNNs, it is also referred to as a Backpropagation-through-time (BPTT) algorithm). In particular, we use a variation of the standard SGD called Adam [9], which allows for adaptive learning rates using the past gradients, similar to using momentum. This results in faster convergence compared to other adaptive algorithms.

User Models	MC	HMM	HsMM
Viterbi	0.6107	0.6107	0.6107
RNN	0	0	0

TABLE III: Comparing error rate of Viterbi coding and RNN method for the toy example. Hardik, please populate the second row with the most recent result.

IV. Experiment

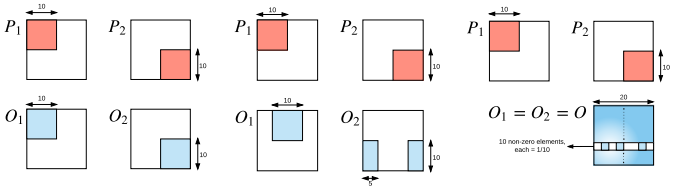
We start with a synthetic sequence generator and test our RNN-based inference algorithm and then move to the real data. For the synthetic experiments, we first compare the performances of the Viterbi-based and RNN-based methods in a toy example and then move to examples with larger state space for detail examination of the RNN-based methods.

A. Viterbi vs. RNN

Here we generate synthetic resolver queue using three different user model of Figure 1 and report the Viterbi and RNN methods performance. To reduce the computational burden for the Viterbi algorithm, we restrict ourselves to 2 pages, 2 users, and 2 possible requests per page. In the MC and HMM user models, this results in an AHMM with 8 hidden states and 2 observation. In the more complicated HsMM user model, there are 32 possible hidden states and 2 observations in the corresponding AHMM. The users shares vector $\boldsymbol{\alpha} = (.4, .6)$ Table III summarizes the result: Interestingly, the performance of the Viterbi is the same through all different user models. Looking at the results reveal that Viterbi report all of the hidden users as user 2 which has the higher share in request generation, i.e., .6. So basically Viterbi can not deinterleave.

Parameter	Value
m	2 users
n	20 pages
q	Maximum of 5 request per page
$\boldsymbol{\alpha}$	$[0.4, 0.6]$
$\mathbf{A}$	Diagonal dominated row stochastic random matrix*.
$\mathbf{P}_u$	A random 20×20 matrix*
$p_w(d)$	Uniform$(1, 5)$
$\mathbf{O}_u$	A random 20×20 matrix*

TABLE IV: Summary of the experimental setup for the synthetic experiment IV-B. *More on the random matrix generation in the text.



(a) Case 1 (b) Case 2 (c) Case 3

Fig. 3: Illustration of the disjoint surfing categories for $a = 10$ and $b = 20$.

B. Synthetic Experiment

All of the results after this part should be re-done using the TensorFlow code. According to the poor performance of AHMM approach from now on we focus on RNN method of Section ?. We report the results of three synthetic experiments: toy example, scaling users, and scaling pages. In all set of experiments, we generate 100 different sequence of length 1000 according to the request generation and interleaving process of Sections ? and ?. The entire data for each instantiation, (100 different sequences) is divided into training, validation and test sets in the ratio 0.6, 0.2 and 0.2 respectively. The model parameters that perform the best on the validation set are saved and using these parameters, the results are reported on the test set. The RNN is trained using the Adam optimizer.

1) Different Browsing Scenarios: In this toy example we test the results for 7 different scenarios, in all of them we want to deinterleave a sequence generated by two users but the parameters in each experiment is set up differently. Table IV specifies the shared parameter setup for the toy example. Specific user transition and emission matrices are set for different scenarios which are explained in Section IV-B1. Note that in our experiments we report results on two set of synthetic data set, where in one we have a users shares vector $\boldsymbol{\alpha}$ determining the share of each user from the queue's requests. In the other more general data generating scheme, we assume that the users transition matrix $\mathbf{A}$ governs the turn in request submission. Different distributions for $\boldsymbol{\alpha}$ and $\mathbf{A}$ are discussed in Section IV-B1.

Sparsity Patterns of Matrices: For each user u we have two matrices $\mathbf{P}_u$ and $\mathbf{O}_u$ which are randomly generated. The generation process assumes that each row of both matrices is sparse, which is a reasonable assumption. Each user view and surf a limited number of pages and on each page the possible requests are from a small subset of the all available pages. The supports of $\mathbf{P}_i$s and $\mathbf{O}_i$s can overlap or be disjoint and this combination generates the different setups of our experiments. After selecting a support we generate a discrete distribution over that support, which will be discussed in Section IV-B1.

In the following the outer-list determines the different strategies for generating $\mathbf{P}_u$s and the inner-list elaborates the method of building $\mathbf{O}_u$s. Each row of $\mathbf{O}_u$s has a non-zero elements (randomly selected) and the distribution is uniform. When call $\mathbf{O}_1 \neq \mathbf{O}_2$ and $\mathbf{O}_1 = \mathbf{O}_2$ schemes, personalized and shared outputs respectively.

- **Disjoint webpage surfing:** In this scenario, users surf disjoint parts of the web, say user 1 surf inside a group of first a pages and user 2 surf the remaining $n-a$ pages, Fig 3.

  Case 1) *Disjoint personalized outputs - same grouping as webpages:* $\mathbf{O}_u$ and $\mathbf{P}_u$ have similar sparsity patterns, Fig 3a.
  Case 2) *Disjoint personalized outputs:* $\mathbf{O}_u$ and $\mathbf{P}_u$ do not have similar sparsity patterns, but support of $\mathbf{O}_1$ and $\mathbf{O}_2$ are disjoint, Fig 3b.
  Case 3) *Shared output:* $\mathbf{O}_1 = \mathbf{O}_2 = \mathbf{O} \in \mathbb{R}^{n \times n}$, Fig 3c.

- **Overlapped webpage surfing with fixed block size:** Each user selects its surfing support of size a at random. Supports may overlap, Fig 4.

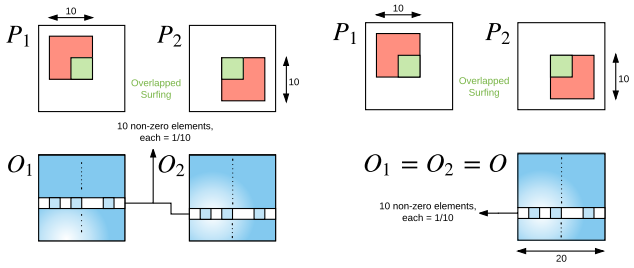
  Case 4) *Personalized outputs:* $\mathbf{O}_1 \neq \mathbf{O}_2 \in \mathbb{R}^{n \times n}$, Fig 4a.
  Case 5) *Shared output:* $\mathbf{O}_1 = \mathbf{O}_2 = \mathbf{O} \in \mathbb{R}^{n \times n}$, Fig 4b.

- **Overlapped webpage surfing with variable block size and interaction between blocks:** Each user selects $s = \text{Uniform}(1, a)$ pages at random as its main support (higher probability of surfing in these s pages), and $a - s$ pages again at random as its auxiliary support (pages that user seldom visits), Fig 5.

  Case 6) *Personalized outputs:* $\mathbf{O}_1 \neq \mathbf{O}_2 \in \mathbb{R}^{n \times n}$, Fig 5a.
  Case 7) *Shared output:* $\mathbf{O}_1 = \mathbf{O}_2 = \mathbf{O} \in \mathbb{R}^{n \times n}$, Fig 5b.

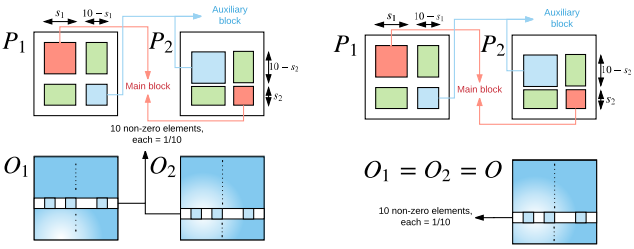
Probability Distributions: Here we explain different distributions used in our synthetic generator:

- Users shares vector $\boldsymbol{\alpha}$: We fix $\boldsymbol{\alpha}$ to $(.4, .6)$.
- Rows of user transition matrix $\mathbf{A}(u)$: This is a diagonal dominant matrix, meaning that if user u has submitted the current request $v(t)$ it is more probable that he submit the next request. In this way, we capture the fact that because of the episodic nature of the request submission, close-by queries are more probable to come from a same user. This has been exploited in the previous literature [**?**]. Note that when instead of matrix $\mathbf{A}$ we only consider the vector $\boldsymbol{\alpha}$ we may not capture this realistic property of the data. For



(a) Case 4 (b) Case 5

Fig. 4: Illustration of the overlapped surfing without auxiliary block for $a = 10$ and $b = 20$.



(a) Case 6 (b) Case 7

Fig. 5: Illustration of the overlapped surfing with auxiliary block for $a = 10$ and $b = 20$.

the toy example, we set $\alpha_{ii} = .5 + \frac{1}{2}\text{Uniform}(0, 1)$ and $\forall i \neq j : \alpha_{ij} \propto (1 - \alpha_{ii})\text{Uniform}(0, 1)$

- Rows of output matrix $\mathbf{O}_u(w)$: As mentioned before, each row $\mathbf{O}_u(w)$ has a non-zero elements with each with probability $1/a$.
- Rows of page transition matrix $\mathbf{P}_u(w)$: In a nutshell, we discretize a continuous Beta distribution with different parameters for each block and normalize the final vector. The distribution that we use for the (main) support is Beta(3+ϵ,1+δ) where ϵ and δ are random numbers from $[-1, 1]$. For the distribution on the auxiliary support of the cases 6 and 7 above, we use Beta(2+ϵ,2+δ).

Discussion: Table V shows the error of our method for all 7 cases of Section IV-B1 for $a = 10$. Note that we report the result for both RNN and its variant LSTM.

Each row is the average result for five instantiation of the model parameters $\mathbf{O}_u$ and $\mathbf{P}_u$. The error of each instantiation (each row) is an average of 100 experiments. Note that case 1 and 2 are trivial cases when both $\mathbf{P}$s and $\mathbf{O}$s are disjoint and LSTM perfectly dis-interleave. Interestingly, performance in case 3 is much worse than cases 1 and 2, which confirms that in our model having disjoint output matrices is more important than disjoint surfing pattern. Intuitively, this makes sense because the final request comes from the output matrices and if we have personalized outputs the deinterleaving should be easier. Interestingly, beyond the trivial cases 1 and 2, case 6 has the best accuracy, probably because of personalized outputs and more complicated $\mathbf{P}_u$ for each user (composed of main and auxiliary block) makes the whole problem more separable.

Cases	Case1		Case2		Case3		Case4		Case5		Case6		Case7	
$\boldsymbol{\alpha}$	R	L	R	L	R	L	R	L	R	L	R	L	R	L
Mean	0	0	0	0	.362	.362	.304	.29	.362	.36	.222	.214	.286	.29
Std	0	0	0	0	.015	.013	.039	.035	.019	.015	.040	.036	.060	.056
A	R	L	R	L	R	L	R	L	R	L	R	L	R	L
Mean	0	0	0	0	.292	.288	.242	.239	.294	.292	.213	.209	.230	.228
Std	0	0	0	0	.057	.053	.058	.054	.105	.101	.057	.057	.056	.052

TABLE V: Deinterleaving error of RNN and LSTM for different cases of the toy example when user transitions are determine by either of $\boldsymbol{\alpha} = [.4, .6]$ or random diagonally dominant $\mathbf{A}$. R and L stand for RNN and LSTM respectively.

Since RNN and LSTM performances are very close and LSTM is more computationally demanding, in the following experiments we will only use RNN method. Note that when we use $\mathbf{A}$ instead of $\boldsymbol{\alpha}$ the error is smaller. This can be explained by the fact that with the diagonally dominated user transition matrix $\mathbf{A}$ neighbor requests in the resolver are more probable coming from a same user and this extra structure will help the algorithm to deinterleave more efficiently.

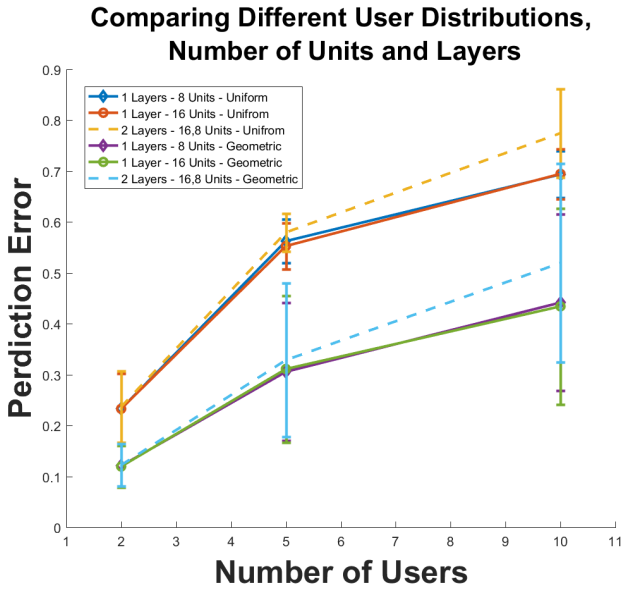
2) Scalability: In this set of synthetic experiment we increase the number of users and number of webpages report the performance of RNN-based deinterleaving methods.

Scaling Users: In this set of experiments we increase the number of users and evaluate the performance of our RNN based method while changing the number of layers and hidden units of RNN. It is expected that adding more users makes the deinterleaving harder and therefore the error will increase. We perform two extreme experiments using each $\boldsymbol{\alpha}$ and $\mathbf{A}$ to control the user transition dynamics:

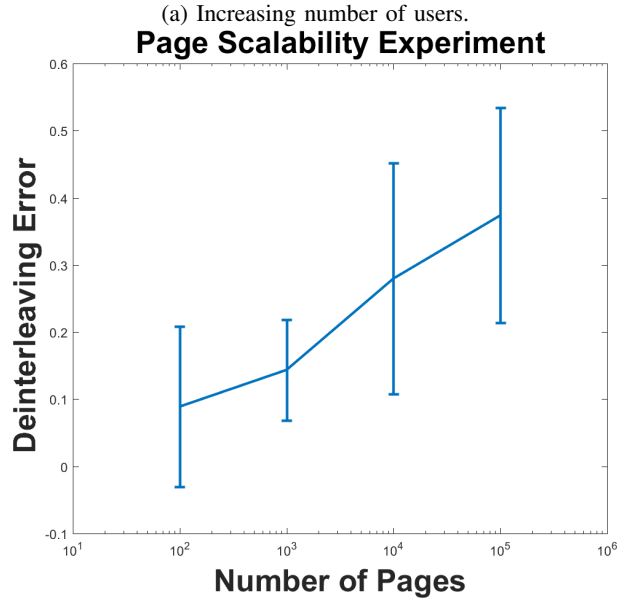
- User transition with $\boldsymbol{\alpha}$:
  - $\boldsymbol{\alpha}$ is uniform over the users, i.e., $\alpha_i = \frac{1}{m}$. In this case, we expect deinterleaving to be very hard because there is no distinction between request generation rate of users.
- User transition with $\mathbf{A}$:
  - $\alpha_{ii} = .5 + \frac{1}{2}\text{Uniform}(0, 1)$ and $\alpha_{ij} = \begin{cases} (1 - \alpha_{ii})(1 - 2^{-m+1})^{-1}.5^j, & j < i \\ (1 - \alpha_{ii})(1 - 2^{-m+1})^{-1}.5^{j-1}, & j > i \end{cases}$. Here, we have two properties that we hope to make deinterleaving easier. First, $\mathbf{A}$ is diagonally dominant which makes close-by queries more probable to come from same users. Also, the overall probability of picking users as active is non-uniform.

We perform experiment on $2, 5$, and 10 users and generate the synthetic data using the Case 6 of Section IV-B1. Maximum number of requests per page is limited to 5 and total number of pages n is 1000 while the support size of rows of matrices a is 100.

Fig 6a illustrates that deinterleaving becomes harder as we increase the number of users. But for the more realistic setup of user transition with matrix A even for 10 users we can reach 50% accuracy. Note that increasing the number of units is not



(a) Increasing number of users.



(b) Increasing number of pages.

Fig. 6: Scalability Experiment.

helping the performance and increasing the layers overfits the data.

Scaling Pages: In this synthetic experiment we test the performance of our algorithm for two users with matrix $\mathbf{A}$ generated as Section IV-B2 while we increase the total number of pages b exponentially while keeping the support of each row of matrices as $a = n/10$. Maximum number of requests per page is limited to 5 and total number of pages b is from the set $\{10^2, 10^3, 10^4, 10^5\}$. Based on the results of the user scalability experiment, we pick single layer RNN with 8 units. Fig 6b shows that as we increase the number of pages denterleaving becomes harder.

C. Real Data

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[2] Tugkan Batu, Sudipto Guha, and Sampath Kannan. Inferring mixtures of markov chains. In *COLT*, volume 2004, pages 186–199. Springer, 2004.

[3] Chris Burge and Samuel Karlin. Prediction of complete gene structures in human genomic dna. *Journal of molecular biology*, 268(1):78–94, 1997.

[4] Christopher B Burge and Samuel Karlin. Finding the genes in genomic dna. *Current opinion in structural biology*, 8(3):346–354, 1998.

[5] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[6] Hongyu Gao, Vinod Yegneswaran, Yan Chen, Phillip Porras, Shalini Ghosh, Jian Jiang, and Haixin Duan. An empirical reexamination of global dns behavior. *ACM SIGCOMM Computer Communication Review*, 43(4):267–278, 2013.

[7] A. Graves and J. Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. In *NIPS*, pages 545–552. 2009.

[8] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[9] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[10] Niels Landwehr. Modeling interleaved hidden processes. In *Proceedings of the 25th international conference on Machine learning*, pages 520–527. ACM, 2008.

[11] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[12] Ariana Minot and Yue M Lu. Separation of interleaved markov chains. In *Signals, Systems and Computers, 2014 48th Asilomar Conference on*, pages 1757–1761. IEEE, 2014.

[13] Gadiel Seroussi, Wojciech Szpankowski, and Marcelo J Weinberger. Deinterleaving markov processes via penalized ml. In *Information Theory, 2009. ISIT 2009. IEEE International Symposium on*, pages 1739–1743. IEEE, 2009.

[14] Gadiel Seroussi, Wojciech Szpankowski, and Marcelo J Weinberger. Deinterleaving finite memory processes via penalized maximum likelihood. *IEEE Transactions on Information Theory*, 58(12):7094–7109, 2012.

[15] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *NIPS*, pages 3104–3112, 2014.